Ceazar Haddad

Content:

1. Introduction:

All the links for videos, articles, websites, and documents, that helped in constructing the code will be included at the end of this document.

1.1. What is the Talespins project:

the following documentation is for clarifying what is the Talespins project and how it was constructed.

Talespins is an LLM (large language model) that generated an article, which contains applications, effects, and benefits of any given topic which of course can be changed depending on the preference of the user, and it also generates 3 images that go with the article. In the end, the results will be presented on a website that was created using the Python package Flask among other packages, which we will cover later on in this documentation.

1.2. Why this topic

This project is part of a project week that aims to teach the students about NLP models (Natural Language Processing models), which is offered by TUM (Technical University of Munich).

To code this we used VS Code, and we created a virtual environment to apply the code in it called Talespins as the project name.

2. General approach:

The main folder with the name Talespins contained another three folders (images, python, templates).

2.1. **Python folder** contains three Python files:

1. generate_images: I used the following packages: openai, base64, os

these packages helped me make the code for generating the images as short and efficient as possible. The prompt that was used for generating the images is the same one the user enters for generating the article in the function 'generate_article' so that the images, as well as the article, would have an obvious relationship so that the user would get something like a web story. The images generated using this function and after calling it using AJAX from the website, will be saved in a folder called images, which we will cover later, and in the HTML code we write the code that goes to the path where these images are saved and then display them on the website.

2. generate_article: I used the following packages: openai, base64, time, os

I tried to make an LLM that takes topics that have applications, effects, and benefits, it does not matter if it was about plants, products, or something like for example artificial intelligence. It will create an article that is divided into these 3 sections.

3. app: I used the following packages: flask, `python.generate_article`, `python.generate_images`

This code is for creating the web application using the Python package flask, and these 2 other packages are basically the ones that I described before. I tried to make them separate so that, if an error occurs or something happens I can find it faster and it makes it easier to modify a specific thing instead of a very long code. So, I imported generate_article from `python.generate_article` and `generate_images from python.generate_images as a function to display the results from these codes on the website.`

`And of course, the flask app included index.html, which goes to a separate folder with the name of templates, which we will talk more about after this one, to generate the look of the website and to present the results.`

2.2. **Templates folder** which contains one file index.html:

This file still has some issues, and I am still working on it, and it is highly likely that they will not be solved by the deadline for delivering this documentation, but I just to give a short description.

In this HTML code I have created 2 buttons (generate article and generate image), so that when the user clicks on generate the article ´, it will take a minute or two and depending on the desired topic that the user wants to know more about the article would be generated and if the user wanted to be on the style of web stories he/she can click on the generate image button and the relevant images will be displayed and the website will take on the form of the web story. This code also includes CSS code in it for the styling and AJAX method to request the Flask app to generate the image and the article.

2.3. **Images folder** which contains the images that were generated by generate_image:

It is located inside the main folder Talespins the same as the others, and its path is included in the HTML code so that the images that are saved in it will be inserted into the website.

3. Code Description:

3.1. generate_article:

in this Python file, we used the API key from OpenAI and then created a function with the name `generate_text(prompt):`

```
def generate_text(prompt):
    response = openai.Completion.create(
        engine="davinci",
        prompt=prompt,
        max_tokens=512,
        n=1,
        stop=None,
        temperature=0.7,
    )
    text = response.choices[0].text.strip()
    return text
```

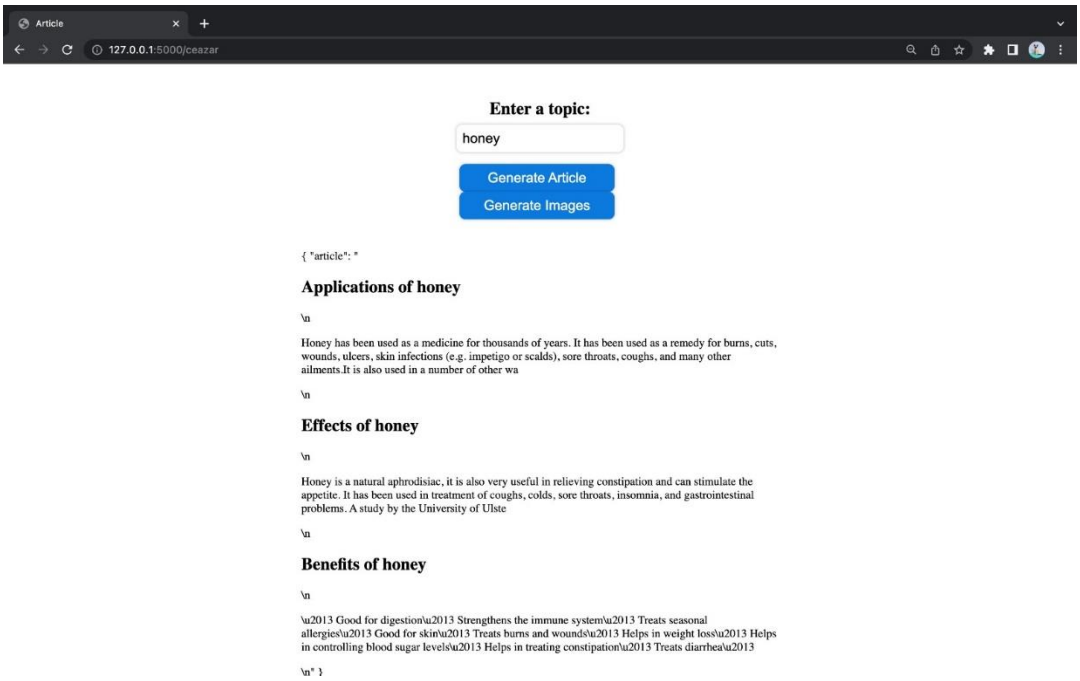

Figure 1 article example with artificial intelligence as keyword

- which as you can see, lays the foundation for generate_article. So basically, it generates text given a prompt and then uses Completion.create, which is a method from openai that sends the prompt to the OpenAI API and gets back a response and the rest of the code determines how the API generates the text. So, we can conclude that the generate_text function is a helper function that takes a prompt and uses the API of OpenAI to generate text for each section of the article and afterward the generate_article function constructs the final article by formatting the generated text into HTML format

  - `text = response.choices[0].text.strip()` → extracts the text from the API response and removes all the whitespaces.

And then we created the other function that we already talked a little about, generate_article(topic) I am going to divide the code to make it easier to explain it:

- ```
  text = generate_text("Applications of " + topic + ":\n")
  applications = text.split("\nEffects of " + topic + ":\n")[0][:250]

  text = generate_text("Effects of " + topic + ":\n")
  effects = text.split("\nBenefits of " + topic + ":\n")[0][:250]

  text = generate_text("Benefits of " + topic + ":\n")
  benefits = text[:250]
  ```

→ the first part, generates the text for each section of the article using the help of generate_text as we mentioned before the method for doing that, and The split function is
  ```

used to split the generated text based on certain keywords, and the resulting substring is truncated to the first 250 characters using the [:250]

- ```
  max_lines = max(applications.count('\n'), effects.count('\n'), benefits.count('\n'))
  applications += '\n' * (max_lines - applications.count('\n'))
  effects += '\n' * (max_lines - effects.count('\n'))
  benefits += '\n' * (max_lines - benefits.count('\n'))
  ```

→ the function pads the sections with empty lines so that they have the same number of lines. This is done using the count method to count the number of newline characters in each section, and then adding empty newline characters to the end of each section as necessary to make them the same length.

- ```
  article = "<h2>Applications of " + topic + "</h2>\n"
  article += "<p>" + applications.replace('\n','') + "</p>\n"
  article += "<h2>Effects of " + topic + "</h2>\n"
  article += "<p>" + effects.replace('\n','') + "</p>\n"
  article += "<h2>Benefits of " + topic + "</h2>\n"
  article += "<p>" + benefits.replace('\n','') + "</p>\n"

  return article
  ```

→ Finally, the function constructs an HTML-formatted article string from the generated sections, with each section header enclosed in a <h2> tag and the section text enclosed in a <p> tag. The resulting article string is returned by the function.

For the construction of this code, I had to go through many trials to get this final version, and I have used ChatGPT to help me in some parts of the codes such as in making the sections have the same size and going deep into the OpenAI documentation and some YouTube tutorials.

3.2. generate_images:

The same as in generate_article I used openai package to make things simpler and I am going to divide this code into 3 different sections to explain:

- ```
  openai.organization = "org-umtU2aMACFq7eNAg7bptz37h"
  ```

→ Sets the OpenAI organization ID to be used for billing and usage tracking. I found out about this after using ChatGPT to modify my code, because I face many issues with generating an image, where I have tried the model that was provided by the LDV
  ```

Lehrstuhl, but I was not able to implement it in the ways I wanted it to be, and it very time consuming

```
def generate_images(topic):
    # Create image using OpenAI API
    res = openai.Image.create(
        prompt=f"{topic}, very render, high resolution",
        n=3,
        size="512x512",
        response_format = "b64_json"
```

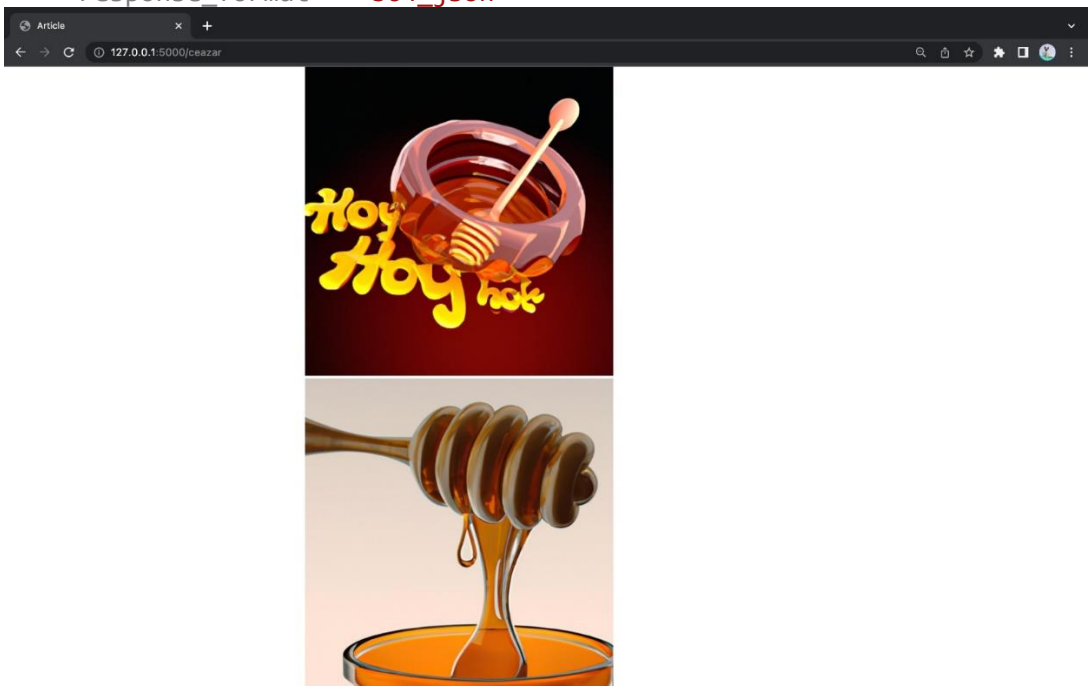

Figure 2 images example with honey as a keyword

→ openai.Image.create(): method to generate an image using the OpenAI API and the prompt is the same topic as in generating the articles code, which we have illustrated many times now.

n = 3: this is the number of images I want the model to generate

and the rest is pretty obvious.

```
for i in range(0, len(res["data"])):
        b64 = res["data"][i]["b64_json"]
        with open(f"Talespins/images/image{i}.jpg", "wb") as f:
            f.write(base64.urlsafe_b64decode(b64))
```

→ This one is a little complicated, so I will explain each line.

- `for i in range(0, len(res["data"])):` Iterates over the number of images generated.

- `b64 = res["data"][i]["b64_json"]`: Extracts the base64-encoded image data from the API response.

- `with open(f"root file/file name/image{i}.jpg", "wb") as f:` root file and file name are to be replaced with the files and folders name you to choose. SO, it opens a file with a name that includes the current image index number (i) and writes the decoded image data to the file.

- `f.write(base64.urlsafe_b64decode(b64))`: it Decodes the base64-encoded image data and writes it to the file opened in the previous line.

I have used this way because the images originally were returning URLs in the terminal and I wanted to save them in a folder on my computer, so I found this way is the most efficient. To help me with this I used OpenAI docs and few YouTube tutorials.

3.3. app:

Let me start by explaining what are the functions used :

1. Flask: is a framework to create applications
2. render_template: is a Flask function used to render HTML templates
3. request: is a Flask object that represents the HTTP request sent by the client.
4. Jsonify: is a Flask function that converts a Python dictionary to a JSON response.

And let me now divide it into 2 separate parts:

```
@app.route('/')
def home():
    return render_template('/index.html')

@app.route('/generate-article')
def generate_article_route():
    topic = request.args.get('topic')

    article = generate_article(topic)

    return jsonify(article=article)
```

→

- `return render_template('/index.html')`: render_template renders index.html, which contains the HTML code we have already said, and it returns the result as an HTTP response.

- `@app.route('/generate-article')`: defines a route, which handles HTTP GET requests that contain a query parameter named topic.

- `request.args.get('topic')`: it extracts the value of topic from the query string.

- `jsonify(article=article)`: the resulting article us returned as a JSON response

```
@app.route('/generate-images')
def generate_images_route():
    topic = request.args.get('topic')


    image_urls = generate_images(topic)
    print(image_urls)

    return jsonify(images=image_urls)
```

Almost the same way as before but for returning the images.

3.4. Index.html

And now let us have a look at the HTML code, I will cover the most important parts of it because there are many lines that contain things like colors, positions, and stuff that are simple to understand if you are not familiar with this project. I am going to divide the code directly by keeping the same order I used in my code.

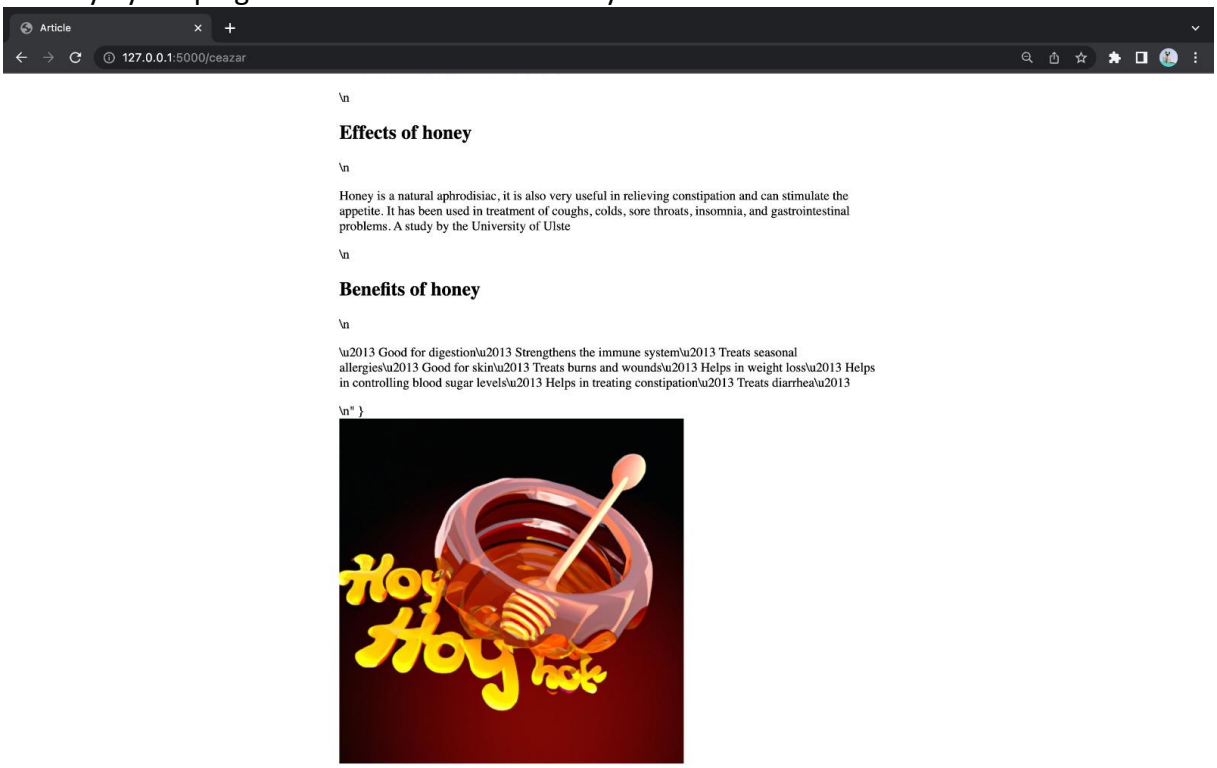

Figure 3 final version (no complete) using the keyword honey

- ```

            <label class="form-label" for="topic-input">Enter a
  topic:</label>
            <input class="form-input" type="text" id="topic-input"
  name="topic">
            <button class="form-submit" type="button"
  onclick="generateArticle()">Generate Article</button>
            <button class="image-submit" type="button"
  onclick="generateImages()">Generate Images</button>

  ```

→This block of code creates a form for the user to input a topic and two buttons for generating an article and generating related images, also it sends an AJAX request to the Flask app to generate an article or related images based on the topic input.

- ```
  <script>
            function generateArticle() {
                var topic = document.getElementById("topic-input").value;

                // Send an AJAX request to the Flask app to generate the
  article
                var xhttp = new XMLHttpRequest();
                xhttp.onreadystatechange = function () {
                    if (this.readyState == 4 && this.status == 200) {
                        document.getElementById("article-
  container").innerHTML = this.responseText;
                    }
                };
                xhttp.open("GET", "/generate-article?topic=" + topic, true);
                xhttp.send();
            }
  ```

→It contains a function generateArticle(), which is triggered when the "Generate Article" button is clicked on the webpage. It first retrieves the value of the "topic" input field on the webpage using document.getElementById("topic-input").value and stores it in a variable called topic. Then, it creates a new instance of the `XMLHttpRequest` object using the `new XMLHttpRequest()` method. This object is used to send a request to the Flask app in order to generate an article based on the topic entered by the user. The function sets up a callback function using xhttp.onreadystatechange, which is triggered when the readyState of the XMLHttpRequest changes (e.g. when the server response is received). In this case, when the readyState is 4 and the status is 200, indicating that the server has successfully responded, the function sets the innerHTML of the article-container div on the webpage to the responseText received from the server, and finally it sends the XMLHttpRequest by calling xhttp.send()

- ```
  function generateImages() {
  ```
  ```

```
                var topic = document.getElementById("topic-input").value;

                // Send an AJAX request to the Flask app to generate the
images
                var xhttp = new XMLHttpRequest();
                xhttp.onreadystatechange = function () {
                    if (this.readyState == 4 && this.status == 200) {
                        var image_data = JSON.parse(this.responseText);
                        var image_urls = image_data.image_urls;
                        var section_html = '<div class="section"><h2
class="section-header">Related Images</h2>';
                        for (var i = 0; i < image_urls.length; i++) {
                            section_html += '<img class="section-image"
src="' + image_urls[i] + '" alt="Related image">';
                        }
                        section_html += '</div>';
                        document.getElementById("article-
container").insertAdjacentHTML('beforeend', section_html);
                    }
                };
                xhttp.open("GET", "/generate-images?topic=" + topic, true);
                xhttp.send();
```

→This is a JavaScript function called generateImages() that is triggered by clicking on the "Generate Images" button. When this function is executed, it sends an AJAX request to the Flask app with the topic value entered by the user in the text input field. Once the Flask app generates the image URLs in response to this request, the generateImages() function receives the JSON data containing the URLs and parses it. Then it iterates over the image URLs in a loop and constructs an HTML string for displaying the images. This string is then inserted into the HTML page using the insertAdjacentHTML() method, which adds the new HTML content at the end of the article-container div.

The result is that this function adds a new section to the page with the related images for the entered topic, if any images are generated by the Flask app.

4. Links:

Links that proved to be useful:

- ChatGPT: https://chat.openai.com/
- OpenAI API: https://platform.openai.com/docs/api-reference?lang=python
- ParametricCamp: https://www.youtube.com/watch?v=q1-7KMHWju4&t=815s